Write a C program to find FOLLOW( ) - predictive parser for the given grammar

S → AaAb / BbBa

A → ∈

B → ∈

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAX 10

char grammar[MAX][MAX];  // Stores the grammar rules
char first[MAX][MAX];    // Stores FIRST sets
char follow[MAX][MAX];   // Stores FOLLOW sets
int ruleCount;           // Number of rules

// Function to check if a character is a non-terminal
int isNonTerminal(char ch) {
   return (isupper(ch));  // A-Z are non-terminals
}

// Function to check if a character is a terminal
int isTerminal(char ch) {
   return (!isupper(ch) && ch != 'ε' && ch != '$');  // Lowercase, digits, symbols
}

// Function to add a symbol to a set (avoid duplicates)
void addToSet(char *set, char symbol) {
   int len = strlen(set);
   for (int i = 0; i < len; i++) {
      if (set[i] == symbol)
         return;  // Avoid duplicates
```

```
    }
    set[len] = symbol;
    set[len + 1] = '\0';
}

// Function to compute FIRST set
void computeFirst(char nonTerminal, char *firstSet) {
    for (int i = 0; i < ruleCount; i++) {
        if (grammar[i][0] == nonTerminal) {  // Find rules for this non-terminal
            char *rhs = &grammar[i][3];  // Right-hand side of production
            if (isTerminal(rhs[0]) || rhs[0] == 'ε') {
                addToSet(firstSet, rhs[0]);  // Add terminal or epsilon
            } else {
                // If RHS starts with a non-terminal, compute FIRST recursively
                char subFirst[MAX] = "";
                computeFirst(rhs[0], subFirst);
                for (int j = 0; subFirst[j] != '\0'; j++) {
                    addToSet(firstSet, subFirst[j]);
                }
            }
        }
    }
}

// Function to compute FOLLOW set
void computeFollow(char nonTerminal, char *followSet) {
    if (nonTerminal == 'S') {
        addToSet(followSet, '$'); // Start symbol: add '$'
    }

    for (int i = 0; i < ruleCount; i++) {
```

```
char *rhs = &grammar[i][3];


for (int j = 0; j < strlen(rhs); j++) {
    if (rhs[j] == nonTerminal) {
        if (rhs[j + 1] != '\0') { // If there is a symbol after non-terminal
            if (isTerminal(rhs[j + 1])) {
                addToSet(followSet, rhs[j + 1]); // Add terminal
            } else {
                char firstNext[MAX] = "";
                computeFirst(rhs[j + 1], firstNext);
                for (int k = 0; firstNext[k] != '\0'; k++) {
                    if (firstNext[k] != 'ε') {
                        addToSet(followSet, firstNext[k]);
                    }
                }
                if (strchr(firstNext, 'ε')) {
                    char followLHS[MAX] = "";
                    computeFollow(grammar[i][0], followLHS);
                    for (int k = 0; followLHS[k] != '\0'; k++) {
                        addToSet(followSet, followLHS[k]);
                    }
                }
            }
        } else { // If non-terminal is at the end
            if (grammar[i][0] != nonTerminal) {
                char followLHS[MAX] = "";
                computeFollow(grammar[i][0], followLHS);
                for (int k = 0; followLHS[k] != '\0'; k++) {
                    addToSet(followSet, followLHS[k]);
                }
            }
```

```
            }
        }
    }
}

// Main function
int main() {
    printf("Enter the number of grammar rules: ");
    scanf("%d", &ruleCount);
    getchar(); // Consume newline

    printf("Enter the grammar rules (in format A->B|c, use 'ε' for epsilon):\n");
    for (int i = 0; i < ruleCount; i++) {
        fgets(grammar[i], MAX, stdin);
        grammar[i][strcspn(grammar[i], "\n")] = '\0';  // Remove newline
    }

    printf("\nFOLLOW Sets:\n");
    for (int i = 0; i < ruleCount; i++) {
        char nonTerminal = grammar[i][0];

        // Check if FOLLOW(nonTerminal) is already computed
        int alreadyComputed = 0;
        for (int j = 0; j < i; j++) {
            if (grammar[j][0] == nonTerminal) {
                alreadyComputed = 1;
                break;
            }
        }
        if (alreadyComputed) continue;
```

```
    char followSet[MAX] = "";
    computeFollow(nonTerminal, followSet);

    printf("FOLLOW(%c) = { ", nonTerminal);
    for (int j = 0; followSet[j] != '\0'; j++) {
        printf("%c ", followSet[j]);
    }
```

Input:

S->AaAb|BbBa

A->ε

Output: